

Apple Assembly Line

Volume 2 -- Issue 2

November, 1981

In This Issue...

Using Applesoft ROMs from Assembly Language	2
Formatted Print Subroutine	6
Poor Man's Disassembler	14
Loops -- A Beginner's Lesson	19

Things For Sale

Here is an up-to-date list of some of the things which I have that you might need: Notice that the prices on books, diskettes, and bags are below retail.

S-C ASSEMBLER II Version 4.0.....	\$55.00
Source code on Disk for above assembler.....	\$95.00
Cross Assembler Patches for 6809 (for 4.0 owners.....	\$20.00
Cross Assembler for 6800/6801/6802 (for 4.0 owners)....	\$22.50
Quarterly Disk #1 (source code from Oct 80 - Dec 80)...	\$15.00
Quarterly Disk #2 (source code from Jan 81 - Mar 81)...	\$15.00
Quarterly Disk #3 (source code from Apr 81 - Jun 81)...	\$15.00
Quarterly Disk #4 (source code from Jul 81 - Sep 81)...	\$15.00
Blank Diskettes (Verbatim, with hub rings, no labels, plain white jackets, in cellophane wrapper).....	20 disks for \$50.00
Zip-lock Bags (2-mil, 6"x9").....	100 bags for \$8.50
Zip-lock Bags (2-mil, 9"x12").....	100 bags for \$13.00
Back Issues of "Apple Assembly Line".....	each \$1.20
"Beneath Apple DOS", Don Worth & Peter Lechner.....	\$18.00
"What's Where in the Apple", William Luebbert.....	\$14.00
"6502 Assembly Language Programming", Lance Leventhal..	\$16.00

I add shipping charges to orders for books and bags. If you are in Texas, remember to add 5% sales tax on books, disks, and bags. Software isn't taxable in Texas.

Advertising Info

If you have a software or hardware product that you want to sell, you can reach over 500 serious Apple owners by advertising in AAL. A full page is only \$20, and a half page \$10. I print 1000 copies, because many orders for back issues come in.

Using Applesoft ROM's from Assembly Language

There are many useful entry points in the Applesoft ROM's. The problem is figuring out how to use them. John Crossley's article "Applesoft Internal Entry Points" (originally published in Apple Orchard Volume 1 Number 1 March 1980) gives a brief description of most of the usable subroutines. If you missed the article, you can still get it from the International Apple Corps. It has also recently been reprinted in "Call A.P.P.L.E. in Depth--All About Applesoft".

Now I want to show you how to use the floating point math subroutines. I won't cover every one of them, but enough to do most of the things you would ever need to do. This includes load, store, add, subtract, complement, compare, multiply, divide, print, and formatted-print.

Internal Floating Point Number Format

Applesoft stores floating point numbers in five bytes. The first byte is the binary exponent; the other four bytes are the mantissa: ee mm mm mm mm.

The exponent (ee) is a signed number in excess-\$80 form. That is, \$80 is added to the signed value. An exponent of +3 will be stored as \$83; of -3, as \$7D. If ee = \$00, the entire number is considered to be zero, regardless of what the mantissa bytes are.

The mantissa is considered to be a fraction between \$.80000000 and \$.FFFFFFFF. Since the value is always normalized, the first bit of the mantissa is always "1". Therefore, there is no need to actually use that bit position for a mantissa bit. Instead, the sign of the number is stored in that position (0 for +, 1 for -). Here are some examples:

-10.0	84 A0 00 00 00
+10.0	84 20 00 00 00
+1.0	81 00 00 00 00
+1.75	81 60 00 00 00
-1.75	81 E0 00 00 00
+1	7D 4C CC CC CD

The Applesoft math subroutines use a slightly different format for faster processing, called "unpacked format". In this format the leading mantissa bit is explicitly stored, and the sign value is stored separately. Several groups of page-zero locations are used to store operands and results. The most frequently used are called "FAC" and "ARG". FAC occupies locations \$9D thru \$A2; ARG, \$A5 thru \$AA.

Loading and Storing Floating Point Values

There are a handful of subroutines in ROM for moving numbers into and out of FAC and ARG. Here are the five you need to know about.

```
AS.MOVFM    $EAF9    unpack (Y,A) into FAC
```

AS.MOVFM	\$EB2B	pack FAC into (Y,X)
AS.MOVFA	\$EB53	copy ARG into FAC
AS.MOVAF	\$EB63	copy FAC into ARG
AS.CONUPK	\$E9E3	unpack (Y,A) into ARG

All of the above subroutines return with the exponent from FAC in the A-register, and with the Z-status bit set if (A)<0.

Here is an example which loads a value into FAC, and then stores it at a different location.

```

LDA #VAR1
LDY /VAR1      ADDRESS IN (Y,A)
JSR AS.MOVFM
LDX #VAR2
LDY /VAR2      ADDRESS IN (Y,X)
JSR AS.MOVFM

```

Arithmetic Subroutines

Once a number is unpacked in FAC, there are many subroutines which can operate on it.

AS.NEGOP	\$EED0	FAC = -FAC
AS.FOUT	\$ED34	convert FAC to decimal ASCII string starting at \$0100
AS.FCOMP	\$EBB2	compare FAC to packed number at (Y,A) return (A) = 1 if (Y,A) < FAC (A) = 0 if (Y,A) = FAC (A) =FF if (Y,A) > FAC
AS.FADD	\$E7BE	load (Y,A) into ARG, and fall into...
AS.FADDT	\$E7C1	FAC = ARG + FAC
AS.FSUB	\$E7A7	load (Y,A) into ARG, and fall into...
AS.FSUBT	\$E7AA	FAC = ARG - FAC
AS.FMUL	\$E97F	load (Y,A) into ARG, and fall into...
AS.FMULT	\$E982	FAC = ARG * FAC
AR.FDIV	\$EA66	load (Y,A) into ARG, and fall into...
AS.FDIVT	\$EA69	FAC = ARG / FAC

Here is an example which calculates VAR1 = (VAR2 + VAR3) / (VAR2 - VAR3).

```

LDA #VAR2      VAR2+VAR3
LDY /VAR2
JSR AS.MOVFM    VAR2 INTO FAC
LDA #VAR3
LDY /VAR3
JSR AS.FADD     + VAR3
LDX #VAR1
LDY /VAR1
JSR AS.MOVFM    STORE SUM TEMPORARILY IN VAR1
LDA #VAR3      VAR2-VAR3

```

```

LDY /VAR3
JSR AS.MOVFM      VAR3 INTO FAC
LDA #VAR2
LDY /VAR2
JSR AS.FSUB       VAR2-VAR3
LDA #VAR1
LDY /VAR1
JSR AS.FDIV       DIVIDE DIFFERENCE BY SUM
LDX #VAR1
LDY /VAR1
JSR AS.MOVMF      STORE THE QUOTIENT

```

As you can see, it is easy to get confused when writing this kind of code. It is so repetitive, there are so many setups of (Y,A) and (Y,X) addresses, that I make a lot of typing mistakes. It would be nice if there was an interface program between my assembly language coding and the Applesoft ROMs. I would rather write the above program like this:

```

JSR FP.LOAD      VAR2 INTO FAC
.DA VAR2
JSR FP.SUB       -VAR3
.DA VAR3
JSR FP.STORE     SAVE AT VAR1
.DA VAR0
JSR FP.LOAD      VAR2 INTO FAC
.DA VAR2
JSR FP.ADD       +VAR3
.DA VAR3
JSR FP.DIV       /(VAR2-VAR3)
.DA VAR1
JSR FP.STORE     STORE IN VAR1
.DA VAR1

```

Easy Interface to Applesoft ROMs

The first step in constructing the "easy interface" is to figure out a way to get the argument address from the calling sequence. That is, when I execute:

```

JSR FP.LOAD
.DA VAR1

```

how does FP.LOAD get the address VAR1?

I wrote a subroutine called GET.ADDR which does the job. Every one of my FP. subroutines starts by calling GET.ADDR to save the A-, X-, and Y-registers, and to return with the address which followed the JSR FP... in the Y- and A-registers. In fact, I return the low-byte of the address in both the A- and X-registers. That way the address is ready in both (Y,A) and (Y,X) form.

GET.ADDR is at lines 4260-4480. I save A, X, and Y in three local variables, and then pull off the return address from the stack and save it also. (This is the return to whoever called GET.ADDR). Then I save the current TXTPTR value. This is the pointer Applesoft uses when picking up bytes from your program to interpret them. I am going to borrow the CHRGET subroutine, so I

need to save the current TXTPTR and restore it when I am finished. Then I pull the next address off the stack and stuff it into TXTPTR. This address is the return address to whoever called the FP... subroutine. It currently points to the third byte of that JSR, one byte before the .DA address we want to pick up.

I next call GET.ADDR2, which uses CHRGET twice to pick up the next two bytes after the JSR and returns them in X and Y. Then I push the return address I saved at the beginning of GET.ADDR, and RTS back. Note that TXTPTR now points at the second byte of the .DA address. It is just right for picking up another argument, or for returning. If there is another argument, I get it by calling GET.ADDR2 again. When I am ready for the final return, I do it by JMPing to FP.EXIT.

FP.EXIT, at lines 4670-4790, pushes the value in TXTPTR on the stack. It is the correct return address for the JSR FP.... Then I restore the old value of TXTPTR, along with the A-, X-, and Y-registers. And the RTS finishes the job.

The Interface Subroutines

I have alluded above to the "FP..." subroutines. In the listing I have shown eight of them, and you might add a dozen more after you get the hang of it.

FP.LOAD	load a value into FAC
FP.STORE	store FAC at address
FP.ADD	FAC = FAC + value
FP.SUB	FAC = FAC - value
FP.MUL	FAC = FAC * value
FP.DIV	FAC = FAC / value
FP.PRINT	print value the way Applesoft would
FP.PRINT.WD	print value with D digits after decimal in a W-character field

FP.LOAD, FP.STORE, FP.ADD, and FP.MUL are quite straightforward. All they do is call GET.ADDR to get the argument address, JSR into the Applesoft ROM subroutine, and JMP to FP.EXIT.

FP.SUB and FP.DIV are a little more interesting. I didn't like the way the Applesoft ROM subroutines ordered the operands. It looks to me like they want me to think in complements and reciprocals. Remember that AS.FDIV performs $FAC = (Y,A) / FAC$. It is more natural for me to think left-to-right, so my FP.DIV permorms $FAC = FAC / value$. Likewise for FP.SUB.

I reversed the sense of the subtraction after-the-fact, by just calling AS.NEGOP to complement the value in FAC. Reversing the division has to be done before calling AS.FDIV. I saved the argument address on the stack, called AS.MOVAF to copy FAC into ARG, called AS.MOVFM to get the argument into FAC, and then called AS.FDIVT.

FP.PRINT, at lines 1830-1930, is also quite simple. I call GET.ADDR to set up the argument address, and AS.MOVFM to load it

into FAC. Then AS.FOUT converts it to an ASCII string starting at \$0100. It terminates with a \$00 byte. A short loop picks up the characters of this string and prints them by calling AS.COUT. I called AS.COUT, rather than \$FDED in the monitor, so that Applesoft FLASH, INVERSE, and NORMAL would operate on the characters.

And now for the really interesting one....

Formatted Print Subroutine

FP.PRINT.WD expects three arguments: the address of the value to be printed, the field width to print it in, and the number of digits to print after the decimal point. Leading blanks and trailing zeroes will be printed if necessary. The Applesoft E-format will be caught and converted to the more civilized form. Fields up to 40 characters wide may be printed, which will accommodate up to 39 digits and a decimal point. If you try to print a number that is too wide for the field, it will try to fit it in by shifting off fractional digits. If it is still too wide, it will print a field of ">>>>" indicating overflow.

For example, look at how values 123.4567 and 12345.67 would be printed for corresponding W and D:

W	D	123.4567	12345.67
10	1	bbbbbb123.4	bbbl2345.6
10	3	bbb123.456	b12345.670
10	5	b123.45670	12345.6700
10	7	123.456700	12345.6700
7	1	bb123.4	12345.6
4	1	123.	>>>>

Sound pretty useful? I can hardly wait to start using it! Now let's walk through the code.

Lines 2380-2410 pick up the arguments. The value is loaded into FAC, and converted to a string at \$0100 by AS.FOUT. Then I get the W and D values into X and Y.

Lines 2420-2510 check W and D. W must not be more than 40; if it is, use 40. (I arbitrarily chose 40 as the limit. If you want a different limit, you can use any value less than 128.) I also make sure that D is less than W. I save W in WD.GT in case I later need to print a field full of ">". Lines 2520-2560 compute W-D-1, which is the number of characters in the field to the left of the decimal point. I save the result back in W.

Lines 2570-2590 check whether AS.FOUT converted to the Applesoft E-format or not. The decimal exponent printed after E is still in \$9A as a binary value. Numbers formatted the civilized way are handled by lines 2600-3160. E-format numbers are restructured by lines 3200-3930.

Lines 2600-2750 scan the string at \$0100 up to the decimal point

(or to the end if no decimal point). In other words, I am counting the number of characters AS.FOUT put before the decimal point. If W is bigger than that, the difference is the number of leading blanks I need to print. Since W is decremented inside the loop, the leading blank count is all that is left in W. But what if W goes negative, meaning that the number is too big for the field? Then I reduce D and try again. If I run out of "D" also, then the field is entirely too small, so I go to PRINT.GT to indicate overflow. If there was no decimal point on the end, the code at lines 2790-2820 appends one to the string.

Lines 2870-2980 scan over the fractional digits. If there are more than D of them, I store the end-of-string code (\$00) after D digits. I also decrement D inside this loop, so that when the loop is finished D represents the number of trailing zeroes that I must add to fill out the field. (If the string runs out before D does, I need to print trailing zeroes.)

At line 3020, the leading blanks are printed (if any; remember that W had the leading blank count). Then lines 3060-3110 print the string at \$0100. And finally, line 3150 prints out D trailing zeroes (D might be zero).

E-formatted numbers are a little tougher; we have to move the decimal point left or right depending on the exponent. We also might have to add zeroes before the decimal point, as well as after the fraction. Lines 3200-3330 scan through the converted string at \$0100; the decimal point (if any) is removed, and an end-of-string byte (\$00) is put where the "E" character is. Now all we have at \$0100 is the sign and a string of significant digits, without decimal point or E-field.

Lines 3350-3600 test the range of the decimal exponent. Negative exponents are handled at lines 3370-3660, and positive ones at lines 3700-3930.

Negative exponents mean that the decimal point must be printed first, then possibly some leading zeroes, and then some significant digits. Lines 3370-3410 compute how many leading zeroes are needed. For example, the value .00123 would be converted by AS.COUT as "1.23E-03". The decimal exponent is -3, and we need two leading zeroes. The number of leading zeroes is $-(\text{dec.exp}+1)$.

There is a little coding trick at line 3370. I want to compute $-(\text{dec.exp}+1)$, and dec.exp is negative. By executing the EOR #\$FF, the value is complemented and one is added at the same time! Why? Because the 6502 uses 2's complement arithmetic. Negative numbers are in the form $256-\text{value}$. EOR #\$FF is the same as doing $255-\text{value}$, which is the same as $256-(\text{value}+1)$. Got it?

Line 3430 prints the leading blanks; lines 3450-3460 print the decimal point. Lines 3480-3520 print the leading zeroes, decrementing D along the way. When all the leading zeroes are out, D will indicate how many significant digits need to be printed.

```

4250 *-----
4260 GET.ADDR
09FB- 8D 36 0A 4270 STA SAVE.A    SAVE A,X,Y REGISTERS
09FE- 8E 37 0A 4280 STX SAVE.X
0A01- 8C 38 0A 4290 STY SAVE.Y
0A04- 68      4300 PLA          SAVE GET.ADDR RETURN ADDRESS
0A05- 8D 35 0A 4310 STA RETLO
0A08- 68      4320 PLA
0A09- 8D 34 0A 4330 STA RETHI
0A0C- A5 B8      4340 LDA AS.TXIPTR SAVE APPLESOFT TEXT POINTER
0A0E- 8D 39 0A 4350 STA SAVE.T
0A11- A5 B9      4360 LDA AS.TXIPTR+1
0A13- 8D 3A 0A 4370 STA SAVE.T+1
0A16- 68      4380 PLA
0A17- 85 B8      4390 STA AS.TXIPTR POINT AT BYTES AFTER JSR FP.<WHATEVER>
0A19- 68      4400 PLA
0A1A- 85 B9      4410 STA AS.TXIPTR+1
0A1C- 20 29 0A 4420 JSR GET.ADDR2 GET FIRST TWO BYTES AFTER
0A1F- AD 34 0A 4430 LDA RETHI    RETURN
0A22- 48      4440 PHA
0A23- AD 35 0A 4450 LDA RETLO
0A26- 48      4460 PHA
0A27- 8A      4470 TXA          ADDR ALSO IN Y,A
0A28- 60      4480 RTS
4490 *-----
4500 GET.ADDR2
0A29- 20 B1 00 4510 JSR AS.CHRGET GET NEXT BYTE IN CALLING SEQUENCE
0A2C- AA      4520 TAX
0A2D- 20 B1 00 4530 JSR AS.CHRGET GET NEXT BYTE IN CALLING SEQUENCE
0A30- A8      4540 TAY
0A31- 60      4550 RTS
4560 *-----
0A32- 4570 W      .BS 1
0A33- 4580 D      .BS 1
0A34- 4590 RETHI   .BS 1
0A35- 4600 RETLO   .BS 1
0A36- 4610 SAVE.A   .BS 1
0A37- 4620 SAVE.X   .BS 1
0A38- 4630 SAVE.Y   .BS 1
0A39- 4640 SAVE.T   .BS 2    TXIPTR
0A3B- 4650 WD.GT    .BS 1
4660 *-----
4670 FP.EXIT
0A3C- A5 B9      4680 LDA AS.TXIPTR+1 GET HIGH BYTE
0A3E- 48      4690 PHA
0A3F- A5 B8      4700 LDA AS.TXIPTR    GET LOW BYTE
0A41- 48      4710 PHA
0A42- AD 39 0A 4720 LDA SAVE.T
0A45- 85 B8      4730 STA AS.TXIPTR
0A47- AD 3A 0A 4740 LDA SAVE.T+1
0A4A- 85 B9      4750 STA AS.TXIPTR+1
0A4C- AD 36 0A 4760 LDA SAVE.A
0A4F- AE 37 0A 4770 LDX SAVE.X
0A52- AC 38 0A 4780 LDY SAVE.Y
0A55- 60      4790 RTS

```

APPLE 8-BIT 8-CHANNEL A/D SYSTEM

- 8-BIT RESOLUTION
- ON BOARD MEMORY- (Just peek at data)
- FAST CONVERSION - (.078 ms per channel).
- ELIMINATES NEED TO WAIT FOR A/D CONVERSION
- A/D PROCESS TOTALLY TRANSPARENT TO APPLE.
- FULL SCALE INPUTS CAN EASILY BE CHANGED BY USER.

APPLIED ENGINEERING'S A/D board is a breakthrough product for all APPLE owners giving real world data at a really affordable price. Diverse applications include monitoring of:

.....TEMPERATURE..... HUMIDITY..... WIND SPEED..... WIND DIRECTION.....
 LIGHT INTENSITY..... PRESSURE..... RPM..... SOIL MOISTURE.....
AND MANY MORE.....

CONTRIBUTED PROGRAMS ARE DISTRIBUTED FREE TO ALL A/D OWNERS IN OUR NEWSLETTER.

See your dealer or contact -

APPLIED ENGINEERING
 P.O. BOX 470301
 DALLAS, TEXAS 75247

\$129

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK

APPLE PERIPHERALS ARE OUR ONLY BUSINESS


```

1000 *
1010 * TEST
1020 *
0800- A0 0A 1030 TEST LDY #10 LOOP 10 TIMES
0802- 20 61 08 1040 JSR FP.LOAD VAR1 = 1.0
0805- 13 E9 1050 .DA AS.ONE
0807- 20 6A 08 1060 JSR FP.STORE
080A- 57 08 1070 .DA VAR1
080C- 20 61 08 1080 JSR FP.LOAD VAR2 = 10.0
080F- 50 EA 1090 .DA AS.TEN
0811- 20 6A 08 1100 JSR FP.STORE
0814- 5C 08 1110 .DA VAR2
0816- 20 61 08 1120 .1 JSR FP.LOAD VAR1=(VAR1+1)/VAR2
0819- 57 08 1130 .DA VAR1
081B- 20 8C 08 1140 JSR FP.ADD
081E- 20 13 E9 1150 JSR AS.ONE
0821- 20 AA 08 1160 JSR FP.DIV
0824- 5C 08 1170 .DA VAR2
0826- 20 6A 08 1180 JSR FP.STORE
0829- 57 08 1190 .DA VAR1
082B- 20 61 08 1200 JSR FP.LOAD VAR2=VAR2-1
082E- 5C 08 1210 .DA VAR2
0830- 20 95 08 1220 JSR FP.SUB
0833- 13 E9 1230 .DA AS.ONE
0835- 20 6A 08 1240 JSR FP.STORE
0838- 5C 08 1250 .DA VAR2
083A- 20 BF 08 1260 JSR FP.PRINT.WD
083D- 57 08 08
0840- 03 1270 .DA VAR1,#8,#3
0841- 20 BF 08 1280 JSR FP.PRINT.WD
0844- 57 08 13
0847- 04 1290 .DA VAR1,#19,#4
0848- 20 48 F9 1300 JSR MON.BLANKS 3 SPACES
084B- 20 73 08 1310 JSR FP.PRINT
084E- 57 08 1320 .DA VAR1
0850- 20 8E FD 1330 JSR MON.CROUT PRINT CARRIAGE RETURN
0853- 88 1340 DEY NEXT TRIP AROUND THE LOOP
0854- D0 C0 1350 BNE .1
0856- 60 1360 RTS FINISHED
0857- 1370 VAR1 .BS 5 MY VARIABLES
085C- 1380 VAR2 .BS 5
1390 *
1400 * ARITHMETIC PACKAGE
1410 *
009A- 1420 AS.FOUT.E .EQ $9A
0093- 1430 AS.TEMP1 .EQ $93 THRU $97
00B8- 1440 AS.TXTPTR .EQ $B8,B9
1450 *
00B1- 1460 AS.CHRGET .EQ $00B1
DB5C- 1470 AS.COUT .EQ $DB5C
E7A7- 1480 AS.FSUB .EQ $E7A7 FAC=ARG-FAC
E7BE- 1490 AS.FADD .EQ $E7BE
E913- 1500 AS.ONE .EQ $E913 CONSTANT 1.0
E97F- 1510 AS.FMUL .EQ $E97F
EA50- 1520 AS.TEN .EQ $EA50 CONSTANT 10.0
EA69- 1530 AS.FDIVT .EQ $EA69 DIVIDE ARG BY FAC
EAF9- 1540 AS.MOVFM .EQ $EAF9
EB21- 1550 AS.MOVIF .EQ $EB21
EB2B- 1560 AS.MOVFM .EQ $EB2B
EB63- 1570 AS.MOVAF .EQ $EB63 MOVE FAC TO ARG
ED34- 1580 AS.FOUT .EQ $ED34
EED0- 1590 AS.NEGOP .EQ $EED0 FAC = -FAC
1600 *
F948- 1610 MON.BLANKS .EQ $F948 PRINT 3 BLANKS
FD8E- 1620 MON.CROUT .EQ $FD8E PRINT CR LF
1630 *
1640 * JSR FP.LOAD LOAD VALUE INTO FAC
1650 * .DA <ADDR OF VALUE>
1660 *
1670 FP.LOAD
0861- 20 FB 09 1680 JSR GET.ADDR IN Y,X AND Y,A
0864- 20 F9 EA 1690 JSR AS.MOVFM
0867- 4C 3C 0A 1700 JMP FP.EXIT
1710 *
1720 * JSR FP.STORE STORE FAC
1730 * .DA <ADDR TO STORE IN>
1740 *
1750 FP.STORE
086A- 20 FB 09 1760 JSR GET.ADDR IN Y,X AND Y,A
086D- 20 2B EB 1770 JSR AS.MOVFM
0870- 4C 3C 0A 1780 JMP FP.EXIT

```

```

1790 *-----
1800 *      JSR FP.PRINT      PRINT VALUE IN FREE FORMAT
1810 *      .DA <ADDR OF VALUE TO BE PRINTED>
1820 *-----
1830 FP.PRINT
0873- 20 FB 09 1840 JSR GET.ADDR
0876- 20 F9 EA 1850 JSR AS.MOVFM
0879- 20 34 ED 1860 JSR AS.FOUT
087C- A0 00 1870 LDY #0
087E- B9 00 01 1880 .1 LDA $100,Y
0881- F0 06 1890 BEQ #2
0883- 20 5C DB 1900 JSR AS.COUT
0886- C8 1910 INY
0887- D0 F5 1920 BNE .1 ...ALWAYS
0889- 4C 3C 0A 1930 JMP FP.EXIT
1940 *-----
1950 *      JSR FP.ADD      FAC = FAC + VALUE
1960 *      .DA <ADDR OF VALUE>
1970 *-----
088C- 20 FB 09 1980 FP.ADD JSR GET.ADDR IN Y,X AND Y,A
088F- 20 BE E7 1990 JSR AS.FADD FAC=ARG+FAC
0892- 4C 3C 0A 2000 JMP FP.EXIT
2010 *-----
2020 *      JSR FP.SUB      FAC = FAC - VALUE
2030 *      .DA <ADDR OF VALUE>
2040 *-----
0895- 20 FB 09 2050 FP.SUB JSR GET.ADDR
0898- 20 A7 E7 2060 JSR AS.FSUB FAC=ARG-FAC
089B- 20 D0 EE 2070 JSR AS.NEGOP FAC=-FAC
089E- 4C 3C 0A 2080 JMP FP.EXIT
2090 *-----
2100 *      JSR FP.MUL      FAC = FAC * VALUE
2110 *      .DA <ADDR OF VALUE>
2120 *-----
08A1- 20 FB 09 2130 FP.MUL JSR GET.ADDR IN Y,X AND Y,A
08A4- 20 7F E9 2140 JSR AS.FMUL FAC=ARG*FAC
08A7- 4C 3C 0A 2150 JMP FP.EXIT
2160 *-----
2170 *      JSR FP.DIV      FAC = FAC / VALUE
2180 *      .DA <ADDR OF VALUE>
2190 *-----
08AA- 20 FB 09 2200 FP.DIV JSR GET.ADDR
08AD- 48 2210 PHA
08AE- 98 2220 TYA
08AF- 48 2230 PHA
08B0- 20 63 EB 2240 JSR AS.MOVAF MOVE FAC TO ARG
08B3- 68 2250 PLA
08B4- A8 2260 TAY
08B5- 68 2270 PLA
08B6- 20 F9 EA 2280 JSR AS.MOVFM
08B9- 20 69 EA 2290 JSR AS.FDIVT
08BC- 4C 3C 0A 2300 JMP FP.EXIT
2310 *-----
2320 *      JSR FP.PRINT.WD  PRINT VALUE WITH W.D FORMAT
2330 *      .DA <ADDR OF VALUE>,$<W>,$<D>
2340 *      D = # OF DIGITS AFTER DECIMAL POINT
2350 *      W = # OF CHARACTERS IN WHOLE FIELD
2360 *-----
2370 FP.PRINT.WD
08BF- 20 FB 09 2380 JSR GET.ADDR ADDRESS OF VALUE
08C2- 20 F9 EA 2390 JSR AS.MOVFM VALUE INTO FAC
08C5- 20 34 ED 2400 JSR AS.FOUT CONVERT TO STRING AT $100
08C8- 20 29 0A 2410 JSR GET.ADDR2 (X)=W, (Y)=D
08CB- E0 29 2420 CPX #41 LIMIT FIELD WIDTH TO 40 CHARS
08CD- 90 02 2430 BCC #14
08CF- A2 28 2440 LDX #40
08D1- 8E 32 0A 2450 .14 STX W # CHARACTERS IN WHOLE FIELD
08D4- 8E 3B 0A 2460 STX WD.GT
08D7- CC 32 0A 2470 CPY W FORCE D<W
08DA- 90 04 2480 BCC .13
08DC- AC 32 0A 2490 LDY W
08DE- 88 2500 DEY
08E0- 8C 33 0A 2510 .13 STY D
08E3- CA 2520 DEX COMPUTE W-D-1
08E4- 8A 2530 TXA
08E5- 38 2540 SEC
08E6- ED 33 0A 2550 SBC D
08E9- 8D 32 0A 2560 STA W
08EE- F0 03 2570 LDA AS.FOUT.E SEE IF E-FORMAT
08F0- 4C 48 09 2580 BEQ .12 NO
08F3- A0 00 2590 JMP E.FORMAT
2600 .12 LDY #0

```

```

2610 *
2620 * SCAN TO "." OR END, DECREMENTING W
2630 *
08F5- B9 00 01 2640 .1 LDA $100,Y SCAN TO END OR DECIMAL POINT
08F8- F0 17 2650 BEQ .2 FOUND END, NO DECIMAL POINT
08FA- C9 2E 2660 CMP #1.
08FC- F0 1D 2670 BEQ .3 FOUND DECIMAL POINT
08FE- C8 2680 INY COUNT STRING LENGTH
08FF- CE 32 0A 2690 DEC W
0902- 10 F1 2700 BPL .1 ...UNLESS TOO MANY DIGITS FOR FIELD
0904- A9 00 2710 LDA #0
0906- 8D 32 0A 2720 STA W NEED NO LEADING BLANKS
0909- CE 33 0A 2730 DEC D BACK UP D IF POSSIBLE
090C- 10 E7 2740 BPL .1 TRY AGAIN
090E- 4C DA 09 2750 JMP PRINT.GT OVERFLOW
2760 *
2770 * APPEND DECIMAL POINT SINCE NONE PRESENT
2780 *
0911- A9 2E 2790 .2 LDA #' PUT DECIMAL POINT BACK ON END
0913- 99 00 01 2800 STA $100,Y
0916- A9 00 2810 LDA #0 END OF STRING CHAR
0918- 99 01 01 2820 STA $101,Y
2830 *
2840 * SCAN TO END, DECREMENTING D
2850 * (PUT EOS AFTER D DIGITS)
2860 *
091B- C8 2870 .3 INY NEXT CHAR
091C- AD 33 0A 2880 LDA D
091F- F0 0B 2890 BEQ .5 NO FRACTIONAL DIGITS
0921- B9 00 01 2900 .4 LDA $100,Y COUNT FRACTIONAL DIGITS TO END
0924- F0 0E 2910 BEQ .6 END OF STRING
0926- C8 2920 INY
0927- CE 33 0A 2930 DEC D
092A- D0 F5 2940 BNE .4 STILL NEED MORE DIGITS
2950 *
092C- A9 00 2960 .5 LDA #0 MAKE EOS
092E- 99 00 01 2970 STA $100,Y
0931- 8D 33 0A 2980 STA D NEED NO TRAILING ZEROES
2990 *
3000 * PRINT LEADING BLANKS AS NEEDED
3010 *
0934- 20 E5 09 3020 .6 JSR LEADING.BLANKS
3030 *
3040 * PRINT CONVERTED STRING
3050 *
3060 * COMES HERE WITH (Y)=0
0937- B9 00 01 3070 .8 LDA $100,Y
093A- F0 06 3080 BEQ .9
093C- 20 5C DB 3090 JSR AS.COUT
093F- C8 3100 INY
0940- D0 F5 3110 BNE .8 ...ALWAYS
3120 *
3130 * PRINT TRAILING ZEROES AS NEEDED
3140 *
0942- 20 ED 09 3150 .9 JSR TRAILING.ZEROES
0945- 4C 3C 0A 3160 JMP FP.EXIT
3170 *
3180 * HANDLE NUMBERS WHICH COME IN E-FORMAT
3190 *
3200 E.FORMAT
0948- A2 00 3210 LDX #0
094A- A0 00 3220 LDY #0
094C- B9 00 01 3230 .1 LDA $100,Y SCAN TO "E", CHANGE TO EOS
094F- C9 45 3240 CMP #1E
0951- F0 0B 3250 BEQ .3
0953- C9 2E 3260 CMP #1. SHUFFLE DIGITS AFTER "."
0955- F0 04 3270 BEQ .2 LEFT ONE POSITION
0957- 9D 00 01 3280 STA $100,X
095A- E8 3290 INX
095B- C8 3300 .2 INY
095C- D0 EE 3310 BNE 1 ...ALWAYS
095E- A9 00 3320 .3 LDA #0 EOS
0960- 9D 00 01 3330 STA $100,X
3340 *
0963- A5 9A 3350 LDA AS.FOUT.E EXP AGAIN
0965- 10 3C 3360 BPL .12 EXP>0
0967- 49 FF 3370 EOR #$FF -(EXP+1) IS # ZEROES
0969- CD 33 0A 3380 CMP D SEE IF MORE THAN WE NEED
096C- 90 03 3390 BCC .4 NO
096E- AD 33 0A 3400 LDA D YES, JUST USE D
0971- AA 3410 .4 TAX

```

```

0972- 20 E5 09 3420 *
3430 JSR LEADING.BLANKS
3440 *
0975- A9 2E 3450 LDA #' DECIMAL POINT
0977- 20 5C DB 3460 JSR AS.COUT
3470 *
097A- A9 30 3480 .7 LDA #'0 ZEROES
097C- 20 5C DB 3490 JSR AS.COUT
097F- CE 33 0A 3500 DEC D REDUCE DIGIT COUNT
0982- CA 3510 DEX
0983- D0 F5 3520 BNE .7 MORE ZEROES
3530 *
0985- A0 00 3540 LDY #0
0987- AD 33 0A 3550 LDA D HOW MANY DIGITS?
098A- F0 0E 3560 BEQ .9 NONE
098C- B9 00 01 3570 .8 LDA $100,Y GET A DIGIT
098F- F0 0C 3580 BEQ .10 OUT OF DIGITS
0991- 20 5C DB 3590 JSR AS.COUT
0994- C8 3600 INY
0995- CE 33 0A 3610 DEC D
0998- D0 F2 3620 BNE .8 MORE
099A- 4C 3C 0A 3630 .9 JMP FP.EXIT
3640 *
099D- 20 ED 09 3650 .10 JSR TRAILING.ZEROES
09A0- 4C 3C 0A 3660 JMP FP.EXIT
3670 *
3680 * E-FORMAT WITH EXP>0
3690 *
09A3- CD 32 0A 3700 .12 CMP W SEE IF ENOUGH ROOM
09A6- B0 32 3710 BCS PRINT.GT FILL FIELD WITH ">"
09A8- AA 3720 TAX
09A9- E8 3730 INX # DIGITS AND TRAILING ZEROES
09AA- 49 FF 3740 EOR $FFF -(EXP+1)
09AC- 6D 32 0A 3750 ADC W COMPUT # LEADING BLANKS
09AF- 8D 32 0A 3760 STA W
09B2- 20 E5 09 3770 JSR LEADING.BLANKS
09B5- B9 00 01 3780 .13 LDA $100,Y PRINT SIGNIFICANT DIGITS
09B8- F0 07 3790 BEQ .14
09BA- 20 5C DB 3800 JSR AS.COUT
09BD- CA 3810 DEX
09BE- C8 3820 INY
09BF- D0 F4 3830 BNE .13 ...ALWAYS
09C1- AD 33 0A 3840 .14 LDA D SAVE TRAILING ZERO CNT
09C4- 48 3850 PHA
09C5- 8E 33 0A 3860 STX D SET UP ZEROES BEFORE "."
09C8- 20 ED 09 3870 JSR TRAILING.ZEROES
09CB- 68 3880 PLA RESTORE REAL TRAILING ZERO CNT
09CC- 8D 33 0A 3890 STA D
09CF- A9 2E 3900 LDA #' PRINT DECIMAL POINT
09D1- 20 5C DB 3910 JSR AS.COUT
09D4- 20 ED 09 3920 JSR TRAILING.ZEROES
09D7- 4C 3C 0A 3930 JMP FP.EXIT
3940 *
3950 * PRINT (WD,GT) GREATER THAN SIGNS (">")
3960 *
3970 PRINT.GT
09DA- A9 3E 3980 LDA #'> OVERFLOW
09DC- AC 3B 0A 3990 LDY WD.GT
09DF- 20 F2 09 4000 JSR PRINT.ACHAR.YTIMES
09E2- 4C 3C 0A 4010 JMP FP.EXIT
4020 *
4030 * OUTPUT (W) LEADING BLANKS
4040 *
4050 LEADING.BLANKS
09E5- A9 20 4060 LDA #$20 BLANK
09E7- AC 32 0A 4070 LDY W # TO PRINT
09EA- 4C F2 09 4080 JMP PRINT.ACHAR.YTIMES
4090 *
4100 * OUTPUT (D) TRAILING ZEROES
4110 *
4120 TRAILING.ZEROES
09ED- A9 30 4130 LDA #'0
09EF- AC 33 0A 4140 LDY D
4150 * FALL INTO PRINT.ACHAR.YTIMES
4160 *
4170 * PRINT (Y) REPETITIONS OF (A)
4180 *
4190 PRINT.ACHAR.YTIMES
09F2- F0 06 4200 BEQ .2 (Y) IS 0, DON'T PRINT ANY
09F4- 20 5C DB 4210 .1 JSR AS.COUT
09F7- 88 4220 DEY
09F8- D0 FA 4230 BNE .1
09FA- 60 4240 .2 RTS

```

Lines 3540-3620 print as many significant digits as will fit in the remaining part of the field (maybe none). Of course, the field might be large enough that we also need trailing zeroes. If so, line 3650 prints them.

What if the exponent was positive? Then lines 3700-3710 see if the number will fit in the field. If not, PRINT.GT will fill the field with ">". If it will fit, then the exponent is the number of digits to be printed. The number of leading blanks will be W-dec.exp-1 (the -1 is for the decimal point). Note that line 3740 complements and adds one at the same time, to get -(exp+1).

Line 3770 prints the leading blanks, if any. Lines 3780-3830 print the significant digits from the string at \$0100. Lines 3840-3890 print any zeroes needed between the significant digits and the decimal point. Lines 3900-3910 print the decimal point, and line 3920 prints the trailing zeroes.

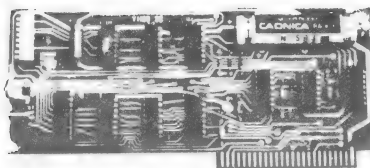
Possible Modifications

You might like to add a dozen or so more FP... subroutines, and hand-compile your favorite Applesoft programs into machine language. You might want to revise the FP.PRINT.WD subroutine to work from Applesoft using the & statement, or using a CALL. This would give you a very effective way of formatting values. You also might want to make it put the result in an Applesoft string variable, rather than directly printing it. You might want to add a floating dollar sign capability, or comma insertion between every three digits. If you implement any of these, let me know. I would like to print them in future issues of AAL.

Time II

The most powerful, easiest to use, clock for your APPLE

- TIME IN HOURS, MINUTES AND SECONDS.
- DATE WITH YEAR, MONTH, DATE, DAY OF WEEK AND LEAP YEAR.
- FAST DATE AND TIME SETTING.
- PROGRAM SELECTABLE 24 HOUR MILITARY FORMAT OR 12 HOUR WITH AM/PM FORMAT.
- ± 30 SECOND ADJUST.
- DIP SWITCH SELECTABLE INTERRUPTS PERMIT FOREGROUND/BACKGROUND OPERATION OF TWO PROGRAMS SIMULTANEOUSLY SO YOU CAN CALL UP SCHEDULES, TIME EVENTS, DATE LISTINGS, AND OTHER PRINTOUTS.
- CRYSTAL CONTROLLED FOR .0005% ACCURACY.
- LATCHED INPUT AND OUTPUT PORTS FOR THE EASIEST PROGRAMMING IN BASIC.
- ON BOARD BATTERY BACKUP POWER FOR OVER 4 MONTHS POWER OFF OPERATION (BATTERY CHARGES WHEN APPLE IS ON).



- INCLUDES 16 SECTOR DISK WITH OVER 25 CONTRIBUTED PROGRAMS SO YOU CAN PUT YOUR TIME II TO USE RIGHT AWAY.
- TWENTY-THREE PAGE OPERATING MANUAL INCLUDED, WITH MANY EXAMPLES OF PROGRAMS TO USE WITH YOUR APPLE IN ANY CONFIGURATION.

ALL ORDERS SHIPPED SAME DAY
SEND \$129.00 CHECK OR MONEY ORDER
(TEXAS RESIDENTS ADD 5% SALES TAX)

APPLIED ENGINEERING
P.O. BOX 470301
DALLAS, TEXAS 75247

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK

APPLE PERIPHERALS ARE OUR *ONLY* BUSINESS

Poor Man's Disassembler.....James O. Church

I wanted a quick and cheap way to get machine language code into the S-C Assembler II Version 4.0, via a text file. I didn't need labels or other automatic features like those \$25-\$30 Two-Pass Disassemblers have. Or at least not badly enough to pay the price and wait for delivery.

There is a fundamental disassembler in the Apple Monitor ROM, which the "L" command invokes. The problems with it are that it only writes on the screen (not on a text file), and it is not in the correct format for the assembler to use. It has too many spaces between the opcode and operand fields, and there is an address rather than a line number at the beginning of each line.

I wrote a program in Applesoft that gets the starting address of the memory you want to disassemble, and then calls on the monitor "L" command as long as you like. The opcode and operand of each disassembled line are packed into a string array until you want to quit. Then you have the option to write the string array on a text file. The program squeezes out the two extra spaces mentioned above, and omits the hex address from each line. In place of the address and blanks which precede the opcode, this program inserts two control-I characters.

Later, when you use EXEC to get the text file into the S-C Assembler II, the first control-I will generate a line number, and the second one will tab over to the opcode column.

To speed it up a little, I wrote a machine language routine to move the second screen line into the string array. I used the last 15 lines of the Field Input Routine from the September, 1981, issue of AAL as a guide. (Thank you, Bob Potts!)

I chose to not use the already overworked "&" way to call my subroutine. Instead I just used CALL 768, followed by the string reference. It works just as well, as far as I'm concerned.

Also, rather than BLOADing such a short little program, I included it as a hexadecimal string inside the Applesoft program. I used an old technique from B. Lam (Call A.P.P.L.E., many moons ago) for passing the hex code to the monitor and thence into memory. (It's all in line 50.)

Line 100 sets up my array for 1280 lines. That's enough for about 2K of code at a time. Plenty. Make it bigger if you like.

Lines 110-120 ask for and process the starting memory address you want. If you type a negative value, I add 65536 to it to make it positive (from 0 thru 65535, rather than -32768 thru 32767). Then I test the range to make sure you ARE in that range.

Line 130 puts the address where the monitor "L" command wants to find it.

The CALL -418 on line 140 disassembles 20 lines. Line 150 shuffles the operand field two spaces left. Then CALL 768A\$(X)

puts the 11-byte string starting with the first character of the opcode on the second screen line, into A\$(X). CALL -912 on line 180 scrolls the screen up one line, so the next line of disassembly is now on the second screen line. The process repeats until 20 lines have been processed.

Then you have the choice to continue or not. If not, you have the option to write A\$() on a text file. If you choose to write it on a file, the file is OPENed, DELETED, OPENed again, and primed for WRITE. Why the DELETE and extra OPEN? So that if the file was already there, it will be replaced with a new one. If a pre-existing file was longer than my new disassembly, the extra old lines would remain in the file.

You know, once the program is in the string array in text form, you could go ahead and scan it for particular addresses in the operand column. Then you could replace them with meaningful symbols. And you could add meaningful labels on lines that are branched to....

[James Church is a special agent for the Northwestern Mutual Life Insurance Agency; he lives in Trumbull, CT. Article ghost-written and program slightly modified by Bob Sander-Cederlof]

NEW UTILITY FOR THE S-C ASM

Do You Like Having Your Labels, Opcodes And Comments Lined Up For Readability, But Don't Want To Give Up Free Format Entry. Or Did You Set Your Tabs For 8 Char Labels And Found You Needed A Few Longer Ones? Now There's No Need To Manually Edit Just To Lineup Those Columns. SC.TAB Is A Source Formatting Utility For Use With The S-C Assembler (4.0). You Simply Specify The Column Numbers For Opcodes And Comments And SC.TAB Does The Rest. A Two-Pass Operation Insures You That There Are No Conflicts Before Any Changes Are Actually Made. Should A Problem Arise Then The Offending Line And The Tab Settings Are Displayed For Your Inspection. The User Can Limit Changes To Any Portion Of Source. SC.TAB Is Fast Since Its 100% Machine Language.

SC.TAB Program Diskette & User Manual: \$15.00

INTRODUCTORY OFFER!

For A Limited Time SC.TAB Is Available At A Special Introductory Price When Purchased With Both Of Our Other S-C Assembler Utilities.

- * SC.XREF : Generates Label Cross Reference Tables For Complete Source Documentation
- * SC.GSR : A Global Search-And-Replace Eliminates Tedious Manual Renaming Of Labels

Normally, All Three Utilities Would Cost \$55.00. Buy Them Now And Save \$10.00

SC Utility Package: \$45.00
(This special offer ends January 1, 1982)

All shipments within continental USA via First-Class mail

Foreign Orders: Add \$3.00 for Air Mail

R A K - W A R E
41 Ralph Road
West Orange NJ 07052

```

40 HOME : VTAB 10: HTAB 9: PRINT "POOR MAN'S DISASSEMBLER": HTAB
   9: PRINT "-----": HTAB 13: PRINT "JAMES O.
   CHURCH": HTAB 14: PRINT "SPECIAL AGENT"
50 HEX$ = "300:20 E3 DF A9 0B 20 52 E4 A0 00 91 83 A5 71 C8 91 83
   A5 72 C8 91 83 A2 94 A0 04 A9 0B 20 E2 E5 60 N D823G": FOR
   I = 1 TO LEN (HEX$): POKE 511 + I, ASC ( MID$ (HEX$,I,1)) +
   128: NEXT : POKE 72,0: CALL - 144
100 DIM A$(1280):X = 0
110 HOME : VTAB 10: INPUT "START LOCATION IN DECIMAL: ";L$:L = VAL
   (L$): IF L < 0 THEN L = L + 65536
120 IF L < 0 OR L > 65535 THEN 110
130 LH = INT (L / 256):LL = L - LH * 256: POKE 58,LL: POKE 59,LH

140 J = 0: HOME : CALL - 418
150 FOR I = 0 TO 6: POKE 1176 + I, PEEK (1178 + I): NEXT
160 CALL 768A$(X)
170 X = X + 1: IF X > 1280 THEN PRINT "ARRAY FULL": GOTO 210
180 CALL - 912:J = J + 1: IF J < 20 THEN 150
190 PRINT : PRINT "CONTINUE? (Y/N) ";: GET A$: IF A$ = "Y" THEN
   140
200 HOME : VTAB 10
210 PRINT "DO YOU WANT TO PUT IT IN A FILE? (Y/N) ";: GET A$: IF
   A$ < > "Y" THEN HOME : END
220 PRINT : INPUT "NAME OF FILE: ";F$
230 D$ = CHR$ (4): PRINT D$"OPEN"F$
240 PRINT D$"DELETE"F$: PRINT D$"OPEN"F$: PRINT D$"WRITE"F$
250 FOR J = 0 TO X - 1: PRINT CHR$ (9); CHR$ (9);A$(J): NEXT
260 PRINT D$"CLOSE": END

```

APPLE MUSIC SYNTHESIZER BREAKTHROUGH

- COMPLETE 16 VOICE MUSIC SYNTHESIZER ON ONE CARD, JUST PLUG IT INTO YOUR APPLE, CONNECT THE AUDIO CABLE (SUPPLIED) TO YOUR STEREO AND BOOT THE SUPPLIED DISK AND YOU'RE READY TO ENTER AND PLAY SONGS.
- IT'S EASY TO PROGRAM MUSIC WITH OUR "COMPOSE" SOFTWARE. YOU'LL START RIGHT AWAY AT INPUTTING YOUR FAVORITE SONGS. OUR MANUAL SHOWS YOU HOW, STEP BY STEP. THE HI-RES SCREEN SHOWS WHAT YOU'VE ENTERED IN STANDARD SHEET MUSIC FORMAT.
- WE GIVE YOU LOTS OF SOFTWARE. IN ADDITION TO "COMPOSE" AND PLAY PROGRAMS, THE DISK IS FULL OF SONGS READY TO RUN.
- FOUR WHITE NOISE GENERATORS (GREAT FOR SOUND EFFECTS).
- PLAYS MUSIC IN TRUE STEREO AS WELL AS TRUE DISCREET QUADRAPHONIC.
- ENVELOPE CONTROL (VOLUME)
- WILL PLAY SONGS WRITTEN FOR ALF SYNTHESIZER (ALF SOFTWARE WILL NOT TAKE ADVANTAGE OF ALL THE FEATURES OF THIS BOARD. THEIR SOFTWARE SOUNDS THE SAME ON OUR SYNTHESIZER).
- AUTOMATIC SHUTOFF ON POWER-UP, OR IF RESET IS PUSHED.
- MANY, MANY MORE FEATURES.

ALL ORDERS SHIPPED SAME DAY
SEND \$159.00 CHECK OR MONEY ORDER
(TEXAS RESIDENTS ADD 5% SALES TAX)

APPLIED ENGINEERING
P.O. BOX 470301
DALLAS, TEXAS 75247

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK
APPLE PERIPHERALS ARE OUR ONLY BUSINESS

:ASM

```

1000 *-----
1010 *      BUILD STRING FROM SECOND LINE ON SCREEN
1020 *-----
1030 *      .OR $300
1040 *-----
DFE3- 1050 PTRGET .EQ $DFE3 PUTS STRING POINTER ADDRESS IN $83,84
E452- 1060 GETSPA .EQ $E452 PUTS ADDRESS OF STRING SPACE IN $71,72
E5E2- 1070 MOVSTR .EQ $E5E2 MOVES DATA FROM (Y,X) TO STRING SPACE
1080 *-----
0071- 1090 SPCPTR .EQ $71,72 PNTR TO STRING SPACE RESERVED BY GETSPA
0083- 1100 STRPTR .EQ $83,84 PNTR TO STRING VARIABLE PTRGET GOT
1110 *-----
1120 *      TO USE:
1130 *      CALL 768A$(X)
1140 *-----
0300- 20 E3 DF 1150 GO JSR PTRGET GET ADDRESS OF STRING INTO $83,84
0303- A9 0B 1160 LDA #11 MOVE 11 BYTES
0305- 20 52 E4 1170 JSR GETSPA GET SPACE FOR 11-BYTE STRING
0308- A0 00 1180 LDY #0
030A- 91 83 1190 STA (STRPTR),Y PUT LENGTH IN STRING DESCRIPTOR
030C- A5 71 1200 LDA SPCPTR LOW BYTE OF STRING ADDRESS
030E- C8 1210 INY
030F- 91 83 1220 STA (STRPTR),Y
0311- A5 72 1230 LDA SPCPTR+1 HIGH BYTE OF STRING ADDRESS
0313- C8 1240 INY
0314- 91 83 1250 STA (STRPTR),Y
0316- A2 94 1260 LDX #S0494 START OF OPCODE ON SECOND SCREEN LINE
0318- A0 04 1270 LDY /S0494 ADDRESS IN (Y,X)
031A- A9 0B 1280 LDA #11 11 BYTES LONG
031C- 20 E2 E5 1290 JSR MOVSTR MOVE IT IN
031F- 60 1300 RTS

```

WHAT, ANOTHER IMPROVEMENT ?

Yes! DISASM The Intelligent Disassembler For The APPLE Has Been Enhanced With More Features Making It One Of The Most Powerful Utilities Of Its Kind. DISASM Converts 6502 Machine Code Into Meaningful, Symbolic Source. The Resultant Text File Can Be Used With Any Of The Most Popular Assemblers. DISASM Is An Invaluable Aid For Understanding And Modifying Machine Language Programs. Here Are The Specs:

DISASM (VERSION 2.2)

- * Selectable output formats are directly compatible with DOS ToolKit, LISA and S-C (4.0) Assemblers. * 100% machine language for fast operation. * Auto-prompting for easy use. * Operates on either the APPLE II or APPLE II Plus. * Labels automatically assigned as Pg Zero, External or Internal. * Labels and addresses are sorted for user convenience. * ORIGIN and EQUATE pseudo-ops provided. * Source segmentation after JMP and RTS allows for easier reading and understanding. * No restriction on disassembled block length (other than RAM or Assembler limitations). * Correctly disassembles displaced object code (The program being disassembled doesn't have to reside in the memory space in which it executes). * User defined Label Name Table replaces arbitrary label assignments (External, Pg Zero and even Internal labels become more meaningful, e.g. JSR COUNT, LDA WNDTOP. The use of the Name Table is optional. * Monitor ROM Label Name Table is included with over 100 of the most commonly used subroutine labels. Label table SOURCE is also provided so you can extend and customize it to your own needs. * Multiple data tables with user defined format may be intermixed with instructions. * NEW ! A FULL Cross-Reference provides a complete table (to screen or printer) grouped by referenced address type. * NEW ! A SINGLE Cross-Reference feature searches through the object code for a single user-specified address.

DISASM (2.2) Program Diskette & User Manual: \$30.00 Upgrade Kit for previous purchasers of DISASM: \$12.50
All shipments within continental USA via First-Class mail Foreign Orders: Add \$3.00 for Air Mail

R A K - W A R E
41 Ralph Road
West Orange NJ 07052

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOL-KIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.

\$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.

\$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.

\$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output forms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.

\$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.

\$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.

\$15 Disk, Applesoft (32K, ROM or Language Card).

(Add \$4.00 for Foreign Mail)

*Apple II is a registered trademark of the Apple Computer Co.

Loops

When you want to program repetitive code in, you write a FOR-NEXT loop or an IF loop. For example, you might write:

```
10 FOR I = 1 TO 10      or: 10 I=0
20 ...                  20 I=I+1 : IF I > 10 THEN 100
30 NEXT I                30 ...
                        40 GO TO 20
                        100
```

How do you do it in assembly language?

Loop Variable in X or Y

One of the simplest kind of loops holds the loop variable in the Y- or X-register, and decrements it once each trip.

```
LOOP   LDY #10      Loop for Y = 10 to 1
.1     DEY
      BNE .1
```

Note that the loop variable is in the Y-register, and that it counts from 10 to 1, backwards. When the DEY opcode changes Y from 1 to 0, the loop terminates.

If you want the loop to execute one more time, with Y=0, change it to this:

```
LOOP   LDY #10      Loop for Y = 10 to 0
.1     DEY
      BPL .1
```

Of course, a loop count of 129 or more would not work with this last example, because Y would look negative after each DEY until the value was less than 128.

If you want the loop variable to run up instead of down, like from 0 to 9, you need to add a comparison at the end of loop:

```
LOOP   LDY #0       Loop for Y = 0 to 9
.1     INY
      CPY #10
      BCC .1        Carry clear if Y < 10
```

All the examples above use the Y-register, but you can do the same thing with the X-register. In fact, using the X-register, you can nest one loop inside another:

```
LOOPS  LDY #0        FOR Y = 0 TO 9
.1     LDX #10       FOR X = 10 TO 1 STEP 1
.2     DEX
      BNE .2        NEXT X
      INY
      CPY #10       NEXT Y
      BCC .1
```

Loop Variable on Stack

Sometimes X and Y are needed for other purposes, and so I use the stack to save my loop variable. Also, the step size can be larger than 1.

```
LOOP   LDA #0      FOR VAR=5 TO 15 STEP 3
.1     PHA          SAVE VAR ON STACK

      PLA          GET VAR FROM STACK
      CLC
      ADC #3        ADD STEP SIZE
      CMP #16
      BCC .1        VAR <= 15
```

In the Apple Monitor ROM there is a double loop using the stack to hold one of the variables. It is used just for a delay loop, with the length of delay depending on the contents of A when you call it. It is at \$FCA8.

```
WAIT   SEC
.1     PHA          outer loop
.2     SEC #1       ...inner loop
      BNE .2        ...next
      PLA
      SEC #1
      BNE .1        next
      RTS
```

The outer loop runs from A down to 1, and the inner loop runs from whatever the current value of the outer loop variable is down to 1. The delay time, by the way, is $5 * A * A / 2 + 27 * A / 2 + 13$ cycles. (A cycle in the Apple II is a little less than one microsecond.)

16-bit Loop Variables

What if you need to run a loop from \$1234 to \$2345? That is a little trickier, but not too hard:

```
LOOP   LDA #$1234   START AT $1234
      STA VARL
      LDA /$1234
      STA VARH

.1     INCL VARL     NEXT: ADD 1
      BNE .2
      INCL VARH
.2     LDA VARL
      CMP $2346
      LDA VARH
      SBC /$2346
      BCC .1        NOT FINISHED
```

A good example of this kind of loop is in the monitor ROMs also. The code for the end of loop incrementing and testing is at \$FCB4-\$FCC8. The memory move command ("M") at \$FE2C-\$FE35 uses this.

Conclusion

There are as many variations on the above themes as there are problems and programmers. Look around in the ROMs, and in programs published in AAL and other magazines; try to understand how the loops you find are working, and adapt them to your own needs.

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$12 per year in the U.S.A., Canada, and Mexico. Other countries add \$12/year for extra postage. Back issues are available for \$1.20 each (other countries add \$1 per back issue for postage). All material herein is copyrighted by S-C SOFTWARE, all rights reserved. Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)